SAFARI Project & Seminars Courses

Exploration of Emerging Memory Systems

Haocong Luo

Prof. Onur Mutlu

ETH Zürich

Fall 2022

11 October 2021

Meeting 2:
Project Ideas

Haocong Luo
Prof. Onur Mutlu
ETH Zürich
Fall 2022
11 October 2021

Projects

1. Extending Ramulator with New DRAM Standards
2. Implementing and Evaluating Hybrid Memory Systems
3. Statistical Extensions For Calculating RowHammer Bit Flip Probability
4. Incorporating RowHammer protection mechanisms in Ramulator
5. Adding Memory Controller-Level Support for In-Memory Computation
6. Making TL-DRAM/SALP Implementations DRAM Standard Agnostic
7. Implementing and Evaluating SARP/DARP
8. Non-volatile Memory Extensions
9. Implement and Evaluating QoS-aware Scheduling Mechanisms
10. Implement and Evaluating Self-Learning Scheduling Mechanisms

Extending Ramulator with New DRAM Standards

Extending Ramulator with New DRAM Standards

Motivation

- Ramulator supports a wide range of DRAM standards, but some recent DRAM standards are not yet available

Goal

- Extend Ramulator with new DRAM standards (e.g., DDR5, LPDDR5, GDDR6)

Expected results

- Modify the memory controller in Ramulator to support new organization and commands that are part of the new DRAM standards

Implementing and Evaluating Hybrid Memory Systems

Implementing and Evaluating Hybrid Memory Systems

Motivation

- Hybrid memory system designs have significant performance and energy efficiency benefits

Goal

- Study existing hybrid memory proposals and implement some of them in Ramulator
- Evaluate the performance and energy efficiency of such design using popular benchmarks

Expected results

- Ramulator with hybrid memory support
- Performance/energy analysis of hybrid memory designs

Related Papers

- Kotra et al., "**CHAMELEON: A Dynamically reconfigurable heterogeneous memory system**", in MICRO' 18.
- Prodromou et al., "**MemPod: A Clustered Architecture for Efficient and Scalable Migration in Flat Address Space Multi-level Memories**", in HPCA' 17.
- Sim et al., "**Transparent Hardware Management of Stacked DRAM As Part of Memory**", in MICRO' 15.
- Agarwal et al., "**Thermostat: Application-transparent Page Management for Two-tiered Main Memory**", in ASPLOS' 17.
- Meswani et al., "**Heterogeneous memory architectures: A HW/SW approach for mixing die-stacked and off-package memories**", in HPCA' 15.
- Lee et al., "**A fully associative, tagless DRAM cache**", in ISCA' 15.
- Vasilakis et al., "**Hybrid2: Combining caching and migration in hybrid memory systems**", in HPCA' 20.
- …

Statistical Extensions to Calculate RowHammer Bit Flip Probability

Estimating RowHammer Bit Flip Probability

Motivation

- Many factors (e.g., access patterns, data pattern dependence, DRAM cell array architecture) affect RowHammer
- Quickly estimating the RowHammer bit flip probability is important for designing new RowHammer attacks and defenses

Goal

- Extend Ramulator to statistically estimate RowHammer bit flip probability during execution of workloads

Expected results

- Ramulator should report RowHammer bit flip probability along with performance statistics

Incorporating RowHammer Protection Mechanisms in Ramulator

RowHammer Protection Mechanisms in Ramulator

Motivation

- Modern DRAM is susceptible to RowHammer
- RowHammer protection mechanisms incur non-negligible performance and energy overheads

Goal

- Implement RowHammer protection mechanisms in Ramulator to evaluate their performance and energy overheads

Expected results

- Support for RowHammer protection mechanisms that can be selectively turned on
- Performance/energy overhead analysis of the implemented RowHammer protection mechanisms

Related Papers

- Kim et al. "**Flipping Bits in DRAM without Accessing Them**," in ISCA' 14.
- Seyedzadeh et al., "**Counter-based Tree Structure for Row Hammering Mitigation in DRAM**", in CAL' 17.
- Son et al., "**Making DRAM Stronger Against Row Hammering**," in DAC' 17.
- You et al., "**MRLoc: Mitigating Row-Hammering Based on Memory Locality**", in DAC' 19.
- Lee et al., "**TWiCe: Preventing Row-hammering by Exploiting Time Window Counters**", in ISCA' 19.
- Park et al., "**Graphene: Strong yet Lightweight Row Hammer Protection**", in MICRO' 20.
- Yaglikci et al., "**BlockHammer: Preventing RowHammer at Low Cost by Blacklisting Rapidly-Accessed DRAM Rows**", in HPCA' 21.
- Marazzi et al., "**ProTRR: Principled yet Optimal In-DRAM Target Row Refresh**", in S&P' 22.

Adding Memory Controller-Level Support for In-Memory Computation

Support for In-Memory Computation

Motivation

- In-memory computing improves execution efficiency by eliminating excessive data movement by moving computation closer to where the data resides

Goal

- Implement easily extensible memory controller-level support for in-memory computation in Ramulator

Expected results

- The memory controller should support simple in-memory operations (e.g., RowClone, Ambit)
- Enabling to easily add new commands for different in-memory operations

Making TL-DRAM/SALP DRAM Standard Agnostic

DRAM Standard Agnostic TL-DRAM/SALP

Motivation

- Ramulator currently implements Tiered-Latency DRAM as a separate DRAM standards
- Same for Subarray-level Parallelism (SALP)

Goal

- Re-implement TL-DRAM and SALP in a DRAM standard agnostic way

Expected results

- Ramulator should implement TL-DRAM and SALP as separate mechanisms
- Using TL-DRAM and SALP in combination with different DRAM standards (e.g., DDR4, LPDDR4) should be supported

Related Papers

- Lee et al., "**Tiered-Latency DRAM: A Low Latency and Low Cost DRAM Architecture**", in HPCA' 13.
- Kim et al., "**A Case for Exploiting Subarray-Level Parallelism (SALP) in DRAM**", in ISCA' 12.

Implementing and Evaluating SARP/DARP

Implementing and Evaluating SARP/DARP

Motivation

- Chang+ [1] show the benefits of parallelizing refreshes with accesses in DRAM

Goal

- Implement SARP and DARP mechanisms in Ramulator

Expected results

- Ramulator extended with SARP/DARP support
- Performance/energy benefits analysis of SARP/DARP using popular benchmark

[1] Chang et al., "Improving DRAM Performance by Parallelizing Refreshes with Accesses", in HPCA' 14.

Related Papers

- Chang et al., "**Improving DRAM Performance by Parallelizing Refreshes with Accesses**", in HPCA' 14.

Non-Volatile Memory Extensions

Non-Volatile Memory Extensions

Motivation

- Emerging Non-Volatile Memory (NVM) technologies offer better device density with DRAM-like access latencies

Goal

- Extend Ramulator to support popular NVM

Expected results

- Model of the performance and endurance of NVM
- Evaluation of popular benchmarks using the implemented NVM

Implement and Evaluating QoS-aware Scheduling Mechanisms

Implement and Evaluating QoS-aware Scheduling Mechanisms

Motivation

- QoS-aware memory request scheduling mechanisms increases overall system performance when dealing with many different workloads with different memory access characteristics

Goal

- Extend Ramulator to support QoS-aware memory scheduling mechanisms

Expected results

- Model and implement QoS-aware memory scheduling mechanism(s).
- Evaluation of the performance of the said mechanism(s).

QoS-Aware Memory Scheduling: Evolution

- Stall-time fair memory scheduling [Mutlu+ MICRO'07]
  - Idea: Estimate and balance thread slowdowns
  - Takeaway: Proportional thread progress improves performance, especially when threads are "heavy" (memory intensive)

- Parallelism-aware batch scheduling [Mutlu+ ISCA'08, Top Picks'09]
  - Idea: Rank threads and service in rank order (to preserve bank parallelism); batch requests to prevent starvation
  - Takeaway: Preserving within-thread bank-parallelism improves performance; request batching improves fairness

- ATLAS memory scheduler [Kim+ HPCA'10]
  - Idea: Prioritize threads that have attained the least service from the memory scheduler
  - Takeaway: Prioritizing "light" threads improves performance

QoS-Aware Memory Scheduling: Evolution

- Thread cluster memory scheduling [Kim+ MICRO'10, Top Picks'11]
  - Idea: Cluster threads into two groups (latency vs. bandwidth sensitive); prioritize the latency-sensitive ones; employ a fairness policy in the bandwidth sensitive group
  - Takeaway: Heterogeneous scheduling policy that is different based on thread behavior maximizes both performance and fairness

- Integrated Memory Channel Partitioning and Scheduling [Muralidhara+ MICRO'11]
  - Idea: Only prioritize very latency-sensitive threads in the scheduler; mitigate all other applications' interference via channel partitioning
  - Takeaway: Intelligently combining application-aware channel partitioning and memory scheduling provides better performance than either

QoS-Aware Memory Scheduling: Evolution

- Parallel application memory scheduling [Ebrahimi+ MICRO'11]
  - Idea: Identify and prioritize limiter threads of a multithreaded application in the memory scheduler; provide fast and fair progress to non-limiter threads
  - Takeaway: Carefully prioritizing between limiter and non-limiter threads of a parallel application improves performance

- Staged memory scheduling [Ausavarungnirun+ ISCA'12]
  - Idea: Divide the functional tasks of an application-aware memory scheduler into multiple distinct stages, where each stage is significantly simpler than a monolithic scheduler
  - Takeaway: Staging enables the design of a scalable and relatively simpler application-aware memory scheduler that works on very large request buffers

QoS-Aware Memory Scheduling: Evolution

- MISE: Memory Slowdown Model [Subramanian+ HPCA'13]
  - Idea: Estimate the performance of a thread by estimating its change in memory request service rate when run alone vs. shared → use this simple model to estimate slowdown to design a scheduling policy that provides predictable performance or fairness
  - Takeaway: Request service rate of a thread is a good proxy for its performance; alone request service rate can be estimated by giving high priority to the thread in memory scheduling for a while

- ASM: Application Slowdown Model [Subramanian+ MICRO'15]
  - Idea: Extend MISE to take into account cache+memory interference
  - Takeaway: Cache access rate of an application can be estimated accurately and is a good proxy for application performance

QoS-Aware Memory Scheduling: Evolution

- BLISS: Blacklisting Memory Scheduler [Subramanian+ ICCD'14, TPDS'16]
  - Idea: Deprioritize (i.e., blacklist) a thread that has consecutively serviced a large number of requests
  - Takeaway: Blacklisting greatly reduces interference enables the scheduler to be simple without requiring full thread ranking

- DASH: Deadline-Aware Memory Scheduler [Usui+ TACO'16]
  - Idea: Balance prioritization between CPUs, GPUs and Hardware Accelerators (HWA) by keeping HWA progress in check vs. deadlines such that HWAs do not hog performance and appropriately distinguishing between latency-sensitive vs. bandwidth-sensitive CPU workloads
  - Takeaway: Proper control of HWA progress and application-aware CPU prioritization leads to better system performance while meeting HWA deadlines

QoS-Aware Memory Scheduling: Evolution

- Prefetch-aware shared resource management [Ebrahimi+ ISCA'11] [Ebrahimi+ MICRO'09] [Ebrahimi+ HPCA'09] [Lee+ MICRO'08'09]
  - Idea: Prioritize prefetches depending on how they affect system performance; even accurate prefetches can degrade performance of the system
  - Takeaway: Carefully controlling and prioritizing prefetch requests improves performance and fairness

- DRAM-Aware last-level cache policies and write scheduling [Lee+ HPS Tech Report'10] [Seshadri+ ISCA'14]
  - Idea: Design cache eviction and replacement policies such that they proactively exploit the state of the memory controller and DRAM (e.g., proactively evict data from the cache that hit in open rows)
  - Takeaway: Coordination of last-level cache and DRAM policies improves performance and fairness; writes should not be ignored

QoS-Aware Memory Scheduling: Evolution

- FIRM: Memory Scheduling for NVM [Zhao+ MICRO'14]
  - Idea: Carefully handle write-read prioritization with coarse-grained batching and application-aware scheduling
  - Takeaway: Carefully controlling and prioritizing write requests improves performance and fairness; write requests are especially critical in NVMs

- Criticality-Aware Memory Scheduling for GPUs [Jog+ SIGMETRICS'16]
  - Idea: Prioritize latency-critical cores' requests in a GPU system
  - Takeaway: Need to carefully balance locality and criticality to make sure performance improves by taking advantage of both

- Worst-case Execution Time Based Memory Scheduling for Real-Time Systems [Kim+ RTAS'14, JRTS'16]

Implement and Evaluating Self-Learning Scheduling Mechanisms

Self-Optimizing DRAM Controllers

- Engin Ipek, Onur Mutlu, José F. Martínez, and Rich Caruana,
  "Self Optimizing Memory Controllers: A Reinforcement Learning Approach"
  *Proceedings of the 35th International Symposium on Computer Architecture (**ISCA**)*, pages 39-50, Beijing, China, June 2008.

Self-Optimizing Memory Controllers: A Reinforcement Learning Approach

Engin İpek[1,2] **Onur Mutlu**[2] **José F. Martínez**[1] **Rich Caruana**[1]

[1]Cornell University, Ithaca, NY 14850 USA
[2] Microsoft Research, Redmond, WA 98052 USA

Projects

1. Extending Ramulator with New DRAM Standards
2. Implementing and Evaluating Hybrid Memory Systems
3. Statistical Extensions For Calculating RowHammer Bit Flip Probability
4. Incorporating RowHammer protection mechanisms in Ramulator
5. Adding Memory Controller-Level Support for In-Memory Computation
6. Making TL-DRAM/SALP Implementations DRAM Standard Agnostic
7. Implementing and Evaluating SARP/DARP
8. Non-volatile Memory Extensions
9. Implement and Evaluating QoS-aware Scheduling Mechanisms
10. Implement and Evaluating Self-Learning Scheduling Mechanisms

SAFARI Project & Seminars Courses

Exploration of Emerging Memory Systems

Haocong Luo

Prof. Onur Mutlu

ETH Zürich

Fall 2022

11 October 2021